Building a music rhythm video game

Ruben Rodrigues Rebelo
Instituto Superior Técnico,
Universidade de Lisboa,
Portugal
E-mail: ruben.rebelo@tecnico.ulisboa.pt

***Abstract*—The video game industry has been growing over the years due to its continuous innovations and creativity. One of the existing video game genres is the music-based video game, in which the gameplay is influenced by the music. Music-based videogames had a irregular path over the years, but nowadays they are expanding not only with traditional gameplay, but with innovative devices that improve the experience.**

In this work, we developed a rhythm music video game prototype, with characteristics from other rhythm music-based games, in which we try to influence the gameplay with the music that is played. For this purpose, we built a game engine to support a particle system and we created a level editor to create music levels. Finally, we evaluated the game experience with users and whether the game achieves the rhythm experience of games of this genre. From such evaluation, we can conclude that our game delivers the essential experience of a music rhythm game.

***Index Terms*—Music, Video Game, Rhythm, Game Development, Level Editor, Game Engine**

I. Introduction

The gaming industry has grown extensively in the last 5 years. According to “2015 Essential Facts About the Computer and Video Game Industry” [3], in 2014, this industry had sold over 135 million video games and generated more than 22 billion dollars in revenue.

Among the different video games genres, there is one called music video games. Music video games are a video game genre where the gameplay is entirely oriented to the interaction of the player actions in synchronization with the music that is played. These actions are normally on the beat of the music. The beat is often defined as the rhythm listeners would tap their toes to when listening to a piece of music.

Over the years, the music-based genre of videogames had an irregular path of evolution. It started on the second half of the 90s by enforcing its presence in Japan, with games like Dance Dance Revolution. During the 2000s it came to the West, with the game franchise Guitar Hero and Rock Band, which included a controller to try to emulate the experience of using a authentic musical instrument and reaching as a popular video game genre in the 2008, behind the action video games.

But it was during the 2009 that this genre had a decline, because the market was saturated by spin-offs from core titles, which led to a nearly 50% drop in revenue [4] , leading to a step back for further expansion in 2010. Despite these setbacks, the music video game market continues to expand, introducing dance-based games that incorporated motion control from Kinect, and Rocksmith, that allows the players to play the songs using a real guitar.

Music-based video games have strong emphasis on music in the game, and the player's enjoyment of the video game is directly related to the specific music that is being played during gameplay. Since player tastes for music vary widely, if a song is enjoyed by one player, it might be unappealing to a majority of other players, making the music preferences fragmented according to the different players.

Playing this type of game is normally synonymous of performing a show. And one of the type of shows that is highly appreciated is the show of fireworks. But among this type of firework shows, the ones where there is music and the music is synchronized with the explosions give a better experience.

The main objective of this work is the development of a music-based video game, where the music or song influences the gameplay with the theme of syncronized fireworks. We also develop a level editor for a easy level creation. The game is build upon a game engine build from scratch, with a particle system for the firework effects.

The rest of this document is structured as follows: first, in section II, we present a short history of music-based video games and an overview of the characteristics of the principles presented in most music video games. In section III, we introduce and discuss some of the related work already done. In section IV we describe the main features that are implemented, in terms of game design. In section V, we give an overview of the technical challenges we faced to implement our engine and building the game. The section VI contains the results and analysis of a playtest in order to validate our work. Finally, in section VII, there is an overall discussion and future work.

II. Background

In this section, we will start by presenting the history of music-based videogames and then we will present some principles of this game genre.

A. History of Music-based Video Games

Before the current music video games for consoles, personal computers and arcade machines, this genre of games started with a electronic toy Simon, built in 1978 by Howard Morrison and by the inventor Ralf Baer. Launched by Studio 54 in New York, Simon had become a cultural icon in the beginning of the 80s, making big success not only in the USA but also in other countries, selling millions of units [6].

In Simon, the player has to press a sequence of randomly generated buttons, each one producing a musical note. For each successful sequence the player answers, a new note is added to the sequence, making it more extensive and requiring a bigger memory effort. It will be that characteristic of pressing the buttons at a given order would became the essence of music video games until today.

At the beginning of the second half of the 80s in Japan, the music-based videogames received a lot of attention by the developers, becoming more and more popular. One game was Otocky (ASCII Corporation, 1987) a 2D shooter for the Famicom System, where a note was played on each of directions the player could fire, making it possible to compose music as the player played the video game [1].

One of the biggest differences in the industry of video games between Japan and the West (America and Europe) was on the popularity of the arcades. While in the USA, the arcades were on the top in the decade of 1970, and today are in decline, in Japan the arcades continue to have a significant presence on the urban landscape. And it was in the arcades that the sequence games of music have became a popular genre and cultivated a financial success in the West in the beginning of the XXI century.

In Beatmania (Konami, 1997), we have control of a similar DJ equipment, turntable included, where the player sees graphic symbols falling in lines, and when those symbols reach the bottom of the screen, they must press the respective buttons to execute a movement.

Konami still continued to evolve their arcade machines. Dance Dance Revolution (Konami, 1998) became a pioneer game in using the body as the controller of the video game. The player uses his feet to press the buttons on a "carpet" following the commands of the screen, and with that, starts to execute dancing movements.

The plastic musical instruments started a commercial success in the West with iconic videogames like Guitar Hero (RedOctane, 2005) and Rock Band (MTV Games, 2007) both developed by Harmonix, where the main controller was a guitar.

But in 2009, the market became saturated with spin-offs from the core titles, which led to a nearly 50% drop in revenue for music game publishers [4] and within a few years, both Rock Band and Guitar Hero series announced they would be taking a hiatus from future titles.

Even with these setbacks, the music videogames continued to expand, introducing a number of dance-based games like Just Dance (Ubisoft, 2009) and Dance Central (2010, MTV Games), that incorporate the use of motion controllers and camera-based controls like Kinect. Also the addition of Rocksmith (2011, Ubisoft), a game with the unique feature of allowing players to plug in virtually any electric guitar and play.

B. *Principles in Music Video Games*

Because adequate metaphors are necessary to build understanding in music-based video games, a grammar was build to constitute the principles for rhythm music-based games [10].

In this analysis of structures of games, the authors concluded the existence of certain features on this genre: active scores, rhythm action, quantisation, synaesthesia, play performance, free-form play and sound agents.

It is important to add that this genre falls between either of three groups: they are rhythm games, electronic instrument games or musical puzzles. In rhythm games the player presses buttons in the provided rhythm and this is progressed by increasing speed and complexity of the rhythms. Later, results are displayed based on performance expressed as a score, allowing competitive play. In electronic instrument games the rhythm - and the melody as well - is generated by the player, the game intends to provide freedom to the player to create his expression. In the puzzle games the player surpasses the challenges by playing a certain musical sequence provided by tunes.

III. Related Work

In this section, we will refer some of the state of the art in music videogames and discuss the influence of music to the gameplay. To finish, we will make a brief discussion, explaining our line of thought to this work.

A. *Osu!*

Osu![1] is a rhythm game originally for Microsoft Windows, developed by Dean "peppy" Herbert in 2007. The gameplay in Osu! is largely similar to a game for the DS "Osu! Tatake! Ouendan!", where a song is played and the player must perform certain actions in accordance with the beat. However , it contains many features that were not in the original game, including multiplayer and also Beatmap (level) creation, among others.

The main objective of the game is to successfully hit enough game objects (hit circles, sliders and spinners) before the health bar depletes to 0 and the song is over. In each level or Beatmap a song is played and the player must hit the game objects at accordance with the rhythm of the song to earn points.

The hit circles are objects that must be hit within the timing associated with the object. When the hit circle appears, a ring of approach starts shrinking around the circle, and the player must click in the circle when the ring is within the border of the circle. The slider appears as two hit circles with a path to follow. When the ring approaches one of the circles, the player hold and follow the path to the other circle, and then release the button. The spinner is a big circle that appears in the middle of the screen, and the player must hold the button and circle the pointer around the screen to earn points.

Every time a successfull hit object is hit, a multiplier is added and is used to calculate the points earned by that hit object. The bigger the multiplier, the more points are earned. If the player misses the game object in time, the multiplier is reset to 0.

[1] https://osu.ppy.sh/

At the end of the level or Beatmap, the player views how many points he earned, how many successfull hits, misses and multiplier he achieved, as well as a statistical graphic of performance. The player earns a grade of performance out of SS, S, A, B, C, D.

The game still provides with a level editor, where players can create their own levels or Beatmaps, and share in the community of the Osu! game. After that any player can download new Beatmaps at the site and play.

The game falls in the rhythm action category. The player must hit the objects in time with the rhythm of the song, and the difficulty level increases by pushing the players limit, by increasing the speed and complexity of the levels.

Also according to the principles of music games it falls in the rhythm game genre, where the player plays the game in reaction to the screen stimulus. At the end of the level, a score of the performance of the player is displayed.

B. Audiosurf

Audiosurf[2] is a puzzle/rhythm hybrid video game created by Invisible Handlebar and released in 2008 for PC. The gameplay of this game mimics a race in a track in which the player must collect blocks to accumulate points and this track is created by the music chosen by the player.

The main feature of this game is the ability to use the player personal songs to generate a level track. Afterwards it is possible to compare scores in an online leaderboard, for the competitive players.

The objective of this video game is to use the ship, controlled by the player, to reach the end of the track by collecting blocks.

In the beginning, after the player selects the song for the track, the level track is generated. The level track has segments uphill, downhill and curvatures for the sides following the attributes from the song. Along with the track, there are blocks that the player must collect to score points. The goal is to collect these blocks and form clusters of 3 or more of the same color. The more blocks accumulated in the cluster, the more points are scored. The blocks in the default settings of the game, appear in hot colors such as red and yellow for more points, or cool colors such as blue and magenta that are worth less.

In each track the player can be awarded medals such as bronze, silver and gold for each difficulty level, awarded at the end upon reaching the total points required.

Sometimes, during gameplay, Overfill may occur. Overfill is caused when the player attempts to add a block to a column which is full and has no blocks forming a match. Doing so will cause the column to empty all blocks, causing a substantial loss of points. It is not possible to cause overfill for about a second after you fill a column up to the top - blocks moved to this column are simply not added.

The player has the ability to choose from 14 different characters distributed between 6 types and 3 difficulty levels, each of which have different abilities that can be performed to aid in the completion of the level.

The category of this game is a musical puzzle, where the player collects blocks and matches them in clusters of three or more blocks of the same color.

The genre falls onto the rhythm action, where the player avoids and collects blocks in synchronization with the rhythm of the music being played.

C. Guitar Hero

The Guitar Hero[3] series is a series of rhythm music games developed by Harmonix in 2005. In this series, this video game is played by using a special controller emulating a guitar.

The video game is played with an extended guitar neck shown vertically on the screen, and as the song progresses, colored markers indicate notes travelling down the screen in syncronization with the music. The note colors and positions match those of the five fret keys on the guitar controller. Once the note(s) reach the bottom, the player must press or play the indicated note(s) by holding down the correct fret button(s) and hitting the strumming bar in order to score points. A Rock Meter changes depending on the sucess or failure caused by the player (denoted by red, yellow, and green sections). If the Rock Meter drops below the red section, the song will automatically end, and the virtual audience boos off the player by his performance.

Player score is incremented for every successfully note hit, and by hitting a long series of consecutive sucessful note hits, the player increases their score multiplier. The game does not focus on accuracy, therefore as long as the player hits each note in the respective window of action, the player receives the same number of points.

Notes can be a single note, or composed of two to five notes that make a chord. Both single notes and chords can also be sustained, indicated by a colored line following the marker. The player can hold sustained note(s) keys down for the entire lenght for additional points. Also, regardless of whether sustains are hit early or late, if the fret is held for the full duration of the hold, the game will always award the same amount of score increase for the note.

Guitar Hero series' categories, in the music-based videogames principles are rhythm action and play as performance. In rhythm action, the player must perform the actions that are shown in the screen and at the end there is a score and performance points depending on the execution of the actions. In play as a performance, the player plays the game in a special guitar that tries to imitate a real guitar, therefore the player can perform like a real guitar rock star.

Guitar Hero falls in the rhythm game genre, where the game button pressing speed and complexity progresses with the speed and complexity of combinations of chords in different difficulty settings.

[2] http://www.audio-surf.com/

[3] https://www.guitarhero.com/

D. Dance Dance Revolution

Dance Dance Revolution[4], abreviated as DDR, is a arcade music video game series produced by Konami only for arcade machines.

The core gameplay involves the player, stepping his or her feet on a “dance platform” to correspond with the arrows that appears on the screen and the beat. During normal gameplay, arrows scroll upwards from the bottom of the screen and pass over a set of stationary arrows near the top (known as the Step Zone). When the scrolling arrows overlap the stationary ones, the player must step on the corresponding arrows on the dance platform, and the player is given a judgement for their accurancy of every note played.

There are different types of arrows. For instance, Freeze Arrows, which is a long green arrow that must be held down until the tail reaches the Step Zone, producing an “O.K.!” judgement if the player succeed or “N.G.” (Not Good) if the player fails to do so, or Shock Arrows, walls of arrows with lightning effects that must avoided, which are scored the same as Freezes (O.K./N.G.); if they are stepped on, a N.G. is awarded, the life bar decreases, and the steps became hidden for a short period of time.

Successfully hitting the arrows in time with the music fills the “Dance Gauge”, or life bar, while failure to do so drains it. If the Dance Gauge is fully depleted during gameplay, the player fails the song, usually resulting in a game over. Otherwise, the player is taken to the Result Screen, which rates the player’s performance with a letter grade and a numerical score, among other statistics.

This video game falls in the rhythm action and play as performance categories. In rhythm action the player must react to screen stimulus, and “dance” in the “carpet” by stepping on the correct positions. In the end there is a score and performance grades acording to the player actions. The player “dances” as a performance of the given inputs percepted.

It is a rhythm game genre, where the complexity increases as the speed of the arrows and arrows sequences complexity increases.

E. Rez

Rez is a rail shooter music videogame released in Japan by Sega in 2001 for Dreamcast and Playstation 2. The game main novelty at the time was the use of sounds and melodies as the player target and destroys foes with electronic music instead of sound effects found in most rail shooters.

The the player takes control of an onscreen avatar travelling along a predetermined path. The player does not control the overall path, only the avatar’s position on the screen. The player targets enemies by holding a “lock-on” button while moving an aiming recule over up to 8 enemies. Once the “lock-on” button is released, the avatar fires shots that target to each target. Failure to hit an enemy or a projectile in time may cause a collision, which reduces the player’s current evolution level by one and changes the avatar form. The game is over if the avatar is hit while at its lowest possible level. At higher evolution levels, the avatar appears as a humanoid figure, while it appears as a pulsating sphere at the lowest level.

Some enemies drop power-up items when destroyed. Two different items enhance the player’s avatar by increasing his/her “evolution bar” by one or three points, respectively. Another item enables the player to trigger an “Overdrive”, which releases a continuous shower of shots at all the enemies of the screen for a short period of time. In some game modes, score bonus items also appear periodically.

Unlike most games, Rez contains almost no sound effects or spoken language. Instead, the game is set to electronic music, which plays in the background and gradually evolves as the player moves along sections. The music is enhanced by musical effects generated by the player’s actions, enemies and surroundings. Player actions are usually locked to the rhythm of the music, such that shots and hits against enemies occur exactly on each beat (as opposed to occuring in real time). Graphical elements such as polygons that make up the player’s avatar, as well as the background elements, also “beat” in time with the music.

The video game falls in the quantisation and synaesthesia categories. In quantisation, the beats of the electronic music, on the background, mark when the exact time when the enemies are hit by the shots fired by the player. In synaesthesia, the background objects of the game vibrate with the beat of the song on the level.

F. Discussion

Nowadays there are a lot of music videogames in the market, and therefore is not possible to study all the games with different gameplay mechanics. The games that were analysed are the more significant in the diversity of gameplay.

Videogames that require special devices similar to a “guitar” of Guitar Hero or a “carpet” to dance as Dance Dance Revolution, with the intention to the player pretend to be a real performer, require a special API and adapters be developed. This kind of API and special adapters may be difficult to find, therefore our solution would not have this type of qualities.

Videogames that create levels with the song from audio analysis (similar to Audiosurf), have strong emphasis in the algorithm, while any type of song could be used (not requiring a dedicated soundtrack). Since the scope of this project is to build a video game, the algorithm could exceed the time required to complete the project (maybe audio analysis could be used as future work).

Games that require a dedicated soundtrack, like Rez, need some work to create such content. Such content can be difficult to come up available or even create.

Osu! type of gameplay is appealing to develop something similar. Simple gameplay, different difficulty levels, content created by players are some of the main points in this video game.

[4] http://www.ddrgame.com/

IV. Design

In this chapter we describe the game design of the project prototype. We will start with the base concept, then describe the gameplay and design of the level editor.

A. Concept

Many or maybe almost all people are fascinated by a show in which the dark sky lights up with little lights that follow a certain dance in the sky: firework show. However when the music is synchronized with the explosions of the fireworks, give an interest effect on people, so this project game is recreating that type of show, but giving the player the control over the show.

The main concept is a game for a touchscreen device in a 2D perspective, where the player executes actions while the song is playing in the background and perform a show of fireworks, actions which are sinchronized with the song being played. Each show is considered a level. In each level, the player explodes fireworks at the rhythm of the song, and depending on his performance, the player receives a score. The performance depends on the timing of a given action of the player.

At the end of the level, there is an overview of the performance of the player in that level, where the plaver can view how many right hits and the misses he had made.

B. Gameplay

The main objective of the game is to explode the fireworks sucessfully, by touching the screen at the right timing, before the firework show enthusiasm meter deplets to 0 or before the song is over. The player explodes fireworks to earn points, depending on the timing for performing the touch. At the end of the level, there is an overview of the score, with the successful explosions and misses, giving a grade to the performance.

The enthusiasm meter is similar to the rock meter from Guitar Hero, and shows how a supposed virtual audience is enjoying the firework show. At the end of the level a performance score is shown. If the enthusiasm meter deplets to 0 before the level ends the is accounted as game over.

The enthusiasm meter is composed by a specific number of chances in which the player can miss or fail the opportunity to attain accuracy on the action performed in a specific context. For every miss or fail, the player loses chances and for every perfect accuracy action he regains chances. This maximum of chances depend on level difficulty.

When a firework is ready to explode, the player can see the position where the firework will explode. Along with the position, a scaled version of the position to press is displayed, shrinking at that specific position (Figure 1). When the two indicators overlap, this indicates the timing to press. Missing this window of time is counted as a miss. It is possible to press the position within this window of action, and the action is awarded a score, depending of how close is to the right timing of the action.

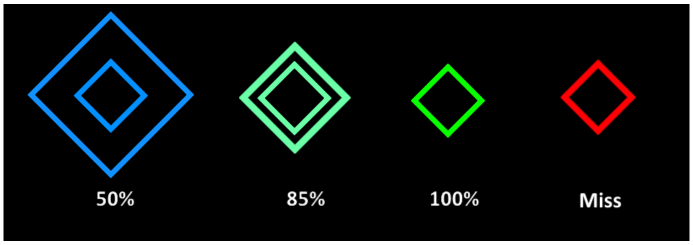


Fig. 1: Timing indicator.

To better acknowledge the right timing, a color is associated with the right timing. The colors work as gradients of the specific phases of the indicator. It starts with blue fading in at a certain limit before the timing, then it shrinks to the right timing to a green color and after this timing it goes to red fading out. This type of identification goes for every action indicator.

There are two types of fireworks: the tail comet and the burst explosion. For every explosion there is a comet that previously starts at the bottom of the screen and performs a path to the specific position in which the firework will explode. When performed at the right timing, in the specific window of time for action, the firework explosion will be performed, or else there is not going to be an explosion, leaving the comet without explosion.

We decided to implement some game objects that perform gameplay interactions throughout the game. These are: simple touch, hold touch with path to follow. Next we will present each interaction in more detail, and afterwards we will discuss the score for each object in more detail.

In the simple touch, a indicator of action will start, only including the timing indicator shown in Figure 1). While the outer edge is approaching the edge of the game object, that we named as note game object, a comet like firework will come from the bottom of the screen and go to the specific point in time to the explosion. If the note object is pressed at the right timing a explosion burst firework is performed. For every value of accuracy above 90% (being 100% the right timing), there is going to be an extra explosion, giving the feedback of a timing touch successfully done.

For the hold touch, the user has to follow a certain path, denoted as slider (Figure 2), which is represented by a proximity indicator and a straight line which the touch must follow. The proximity indicator starts to shrink and after reaching the right position it starts moving automatically on the path. Since the button is small and it is difficult to know where the indicator is moving and at which speed, and the touch must follow, it includes an orange edge large enough to be visible for the player to know where to follow. If the player starts pressing while the object is already following the path, it is counted, but with a decrease in score. This game object is shown comets coming from a path to a specific direction. Firework comets will be generated along the path.

or the feedback of the actions, we decided to have an indicator that the action was performed in accordance for the

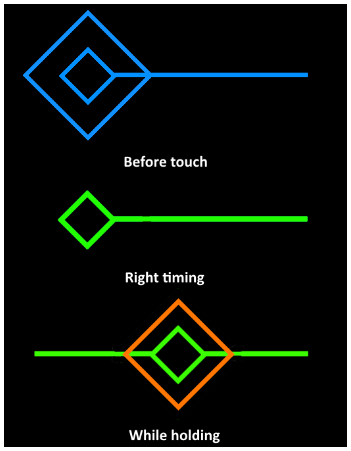


Fig. 2: Slider game object.

different game objects. This feedback appears after the action is performed or missed.

For the note game object a edge equal to the game object will have a color determined by the accuracy: below 70% of accuracy (a miss) is represented by red color, between 70% and 90% ("OK") accuracy is represented by a blue color, and finally above 90% (Perfect) accuracy is not represented, since in this accuraccy level, the main explosion and an extra burst occur. The burst's color is different from the one in the main explosion. Missing the opportunity is represented by red color. This feedback gradually disappears over a very small time.

The slider game object is represented by the same feedback presented previously at the end of the slider. It is a mean value of accuracy between the first touch and the release touch. The first touch is in accordance with the exact time of touch, and the release takes into consideration the exact time of release. This means the better accuracy mean value between this two exact times, the better it is the performance. At the end of this action a edge equal to the to the center of the slider game object will have a color determined by the accuracy with the same color feedback presented previously. Since the slider appears comets coming from the line of the slider, if the player does not follow the slider correctly, new comets will not be generated.

Along with the performance points there is a combo counter that represents the successful consecutive touches that the player performed. This value is going to be used to the score of the successful hit of the objects. This means that each time a miss is performed, the counter is reset to 0 and each sucessfull hit, increments the countes's value by one.

The score points are counted by each suceessful action performed. Missing an action will be not give points. The accuracy value will be multiplied by 50 if the accuracy was inside the OK window, or multiplied by 100 if the accuracy is bigger than 90%. Also each instant this score points is multiplied by the combo counter at that giving time. This means that bigger accuraccy and bigger the combo counter more points earned.

At the end of the level, there is a screen that shows the performance that the player achieved in that level. The score screen contains, the number of misses, the number of "OKs" timings, the number of "Perfect" timings, the final score, the accuracy on all level and a Grade letter of that level performance. Grades can assume values SS, S, A, B, C, D, E.

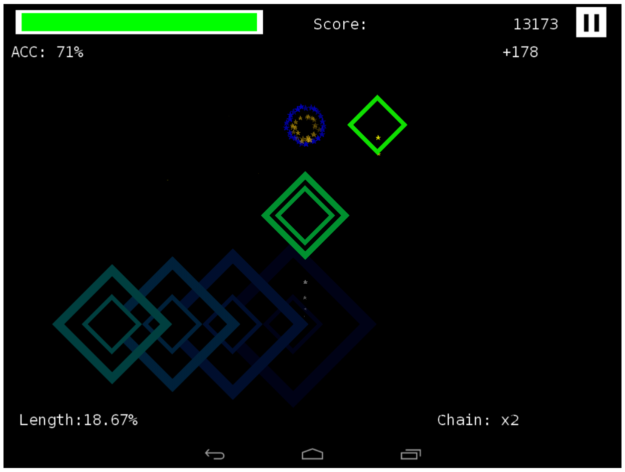


Fig. 3: Screen of gameplay.

C. Level Editor

To allow the creation of the levels to be played, this game needs a level creator. In this section we will describe the various important functionalities that the level editor features.

The level editor has two main views: the editor view and the visualizer. In the editor view, the user can place and edit the actions, and edit the fireworks of the level. The visualizer view can have a preview of how the level would like on the level player.

Figure 4 shows the level editor with the editor view.

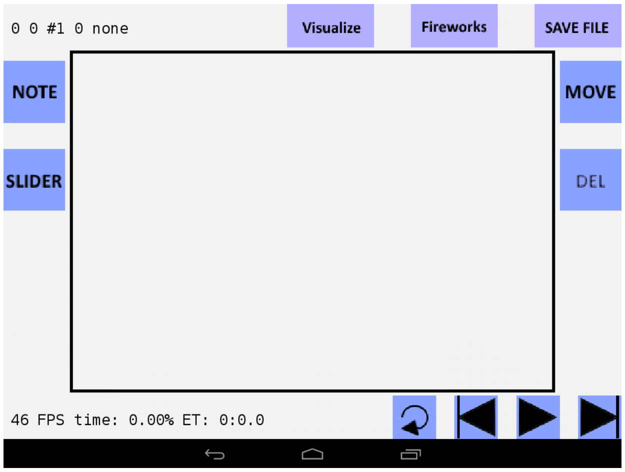


Fig. 4: Level Editor screen.

In any view the level can be saved by pressing the save file button at the top right corner of the editor. It level is saved to a file that contains the song name and the different game objects and fireworks assocciated with them.

The control of the levels works in the timeline of the song. Therefore the user needs to go forward and backwards in the timeline to progress in the level. The game objects appear at timings they are assigned to (fading in before time and fading out after the time) or remain invisible instead.

The first and the most important controls of the editor are the sound controls. The sound controls make it easy to navigate between the timeline of the song. There is the Play button, that plays the song, and pauses it if pressed again. Pressing Play gets the timeline to start moving in real time, made possible by the system clock. Then there are controls to navigate backwards or forward in the timeline.

Next to the controls there is the timeline information of the song: the time and the information of the progress of the song with in a percentage.

In the editor view there are two modes of edition: level mode and fireworks mode. The first mode is for the placement of game objects (notes and sliders) on the timeline. The other mode is to edit the fireworks associated with the game objects. In both views the timeline controls of the song and the information of the time of the sound are kept in the same position.

1) Level Mode: In the Level Mode can place the game objects: the notes and slider game objects. Instances of these objects can be placed in the timeline. The placement timing of the game objects is made in accordance with the song time that is played.

At the right side of the area to place the game objects, there are two buttons to edit the game objects placed in the timeline: the move button and the delete button.

2) Fireworks Mode: After the placement of the game objects, the fireworks associated with the game objects can be edited. The fireworks entity is a group compose of a tail and an explosion. The tail appears 3 seconds before the timing of the explosion, and the explosion effect is played after the timing.

In the firework editor scene, the game object is represented by the firework explosion point and by a trail of dots (the path followed by the firework until reaching the explosion point). When the explosion point is pressed, one can edit the firework effect associated with that particular game object.

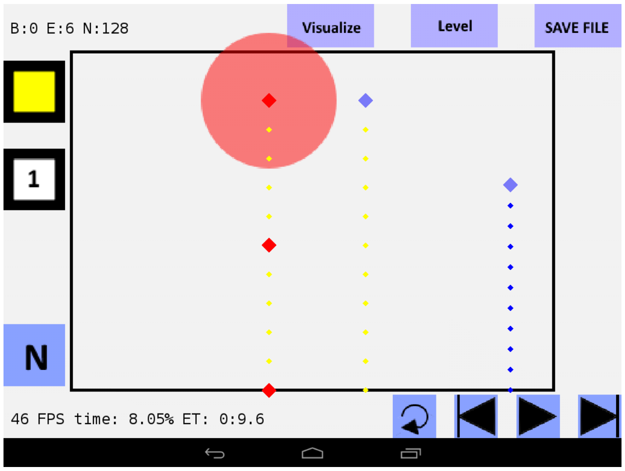
It is possible to change the color of the firework tail and explosion, type of explosion and size of explosion. The tail firework trajectory can also be changed, making curves as trajectories.

For the note game object is associated a tail and explosion firework. And the slider is associated a variable number of comet-like fireworks with configurable functionalities as the tail firework.

It is possible to visualize the preview of the level in the visualizer mode.

V. Implementation

In this section we are going to describe the various steps that we took to construct our project. We will also describe the various challenges that we faced and the solutions that we chose to make a playable prototype of our project.


B:0 E:6 N:128

46 FPS time: 8.05% ET: 0:9.6

Fig. 5: Level Editor firework screen with an editable firework note game object.

Since there was no viable option for an engine for our target device, we decided to build our own engine from scratch.

Instead of using Android SDK, we decided to use Android NDK, (standing for Android Native Development Kit), which contains a set of libraries in C and C++ language. However in this native library are not available the necessary libraries for render a image into a 2D texture, render text from a given string and playback sound from file. So we had to develop such funtionalities.

A. Architecture of the Engine

In this subsection we present the architecture of the engine we built for this project.

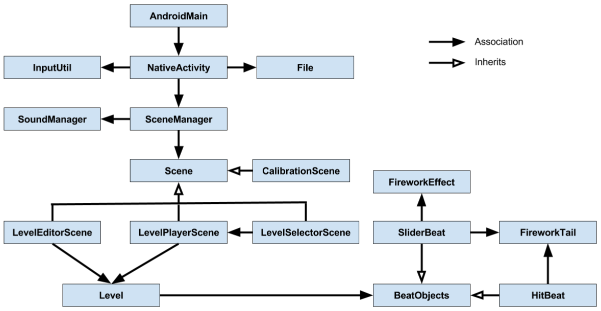


Fig. 6: Engine Architecture.

The components of this game engine are:

- **AndroidMain** function, is the main function that is called when the application starts for the Android environment. The **NativeActivity** class that implements an Android activity and it reads files from the asset folder with the **File** class and read the inputs from the device with the **InputUtil** class.
- For better management of scenes we decided to build a struture of scenes done by the **SceneManager** class and

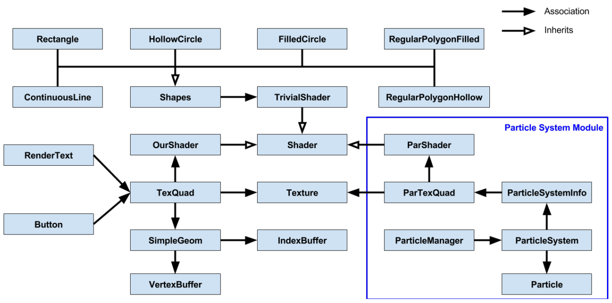


Fig. 7: Helper classes.

Scene class. It represents the main game loop as a simple main loop described in [9].

- The **Texture** class is responsible for displaying images using the libpng[5]. For displaying images we create geometry, pass the images into textures and render the geometry with the textures, functionality in the **TexQuad** that uses a generated **SimpleGeom** mesh in specific classes **VertexBuffer** and **IndexBuffer**, using the **Texture** into it.
- To render text into the screen, in **RenderText** class, we decided to use a external library, FreeType[6], which makes it possible to read font files and use this information to render text. It loads the font file characters into a bitmap and then creates **TexQuads** for each character.
- To play sounds we need to load and play files from the asset folder and/or from the storage drive of the device. For this we use a internal standard embedded hardware-acelerated audio API in Android that was possible to connect via Android NDK: OpenSL ES [5].
- To provide fast recreation of a button input in the touchscreen, there is a **Button** class.
- To render the game objects we generate the geometry in the **Shapes** class. The available shapes are: **HollowCircle**, **FilledCircle**, **Rectangle**, **RegularPolygonFilled**, **RegularPolygonHollow** and **ContinuousLine**.
- Then there is a **ParticleSystem**, which contains an array of particles from **Particle** in memory. The Particle System properties are stored in a structure called **ParticleSystemInfo**. A **ParticleManager** manages the particles, it controls the creation, the render, the update and the deletion of particle systems.

B. Particle System

We made a testing scene were we were able to create to a maximum of **a range of 45 to 50 particles systems** of this kind, which make a total maximum of **12500 particles** (50 systems, 25 particles generated in main particle system and 10 particles for each tail particle system) in an instance. We achieved a range of 17 to 21 FPS (Frames per Second) drop, from a constant of 60 FPS. It is important to refer that every Android device has a VSYNC functionality [7] and maintains the maximum of 60 FPS. The testing scene only ran with particles systems.

C. Challenges

Next we will present two of the challenges that we faced while developing the Scenes: **LevelEditorScene**, **LevelCalibration** and **LevelPlayer**.

1) Time Elapsed: uring this phase we faced a problem: the time elapsed in miliseconds had different values from the time elapsed in the sound file of the sound manager. This is because the sound file from the OpenSL ES pointer of the current position of the song does not progress at the same pace as the date time of the device. The sound pointer of the current position of the song is updated without a constant value from the background thread playing the sound file. To solve this problem we devised the following solution: every time the sound pointer position was updated, the time elapsed from the song was also updated to accomodate the sound position and the time elapse, by the mean value between these two values.

2) Render Calibration: Because of the latency of the device, we made a device calibration scene, in which we made a visual indicator like a metronome, and we asked to tap along with the indicator, calibrating the video lag, like is said in [2]. There was a countdown from 0 to 4, in loop, and when 0 was displayed, a square image would appear, and the user was to press at that time. In this scene there was visible a value of the difference between the exact time and the time pressed, giving the latency for the device. This value was the mean value of the actual difference and the previous difference in time. After many trials, and if the value did not change, that meant that the value was the lantency. Then we add this lantency to the time for the rendering of the objects.

VI. Results

In this section, we present the procedures and results of evaluating the prototype.

1) Playtesting: For the playtesting we created two levels: The first level song chosen was Justin Timberlake's "Can't Stop The Feeling" and the second level song chosen was "Sia's The Greatest".

Participants tested our game in a room, with no distractions. We started by explaining that they would play 3 levels of a prototype of a game with a basic description, and asked to answer a questionnaire and answer some open questions as a interview. They were asked to start by responding the first part of the questionnaire with some questions, and then they played a level twice. One time with the song playing the background and another without the song, alternating the order for each participant, to assess if playing with music would increase the score, therefore the experience. Next, they would answer another part of the questionnaire, and play a third time, and afterwards they would do the rest of the questionnaire.

The questionnaire was anonymous. The questionnaire started with an assessment of the player profile. Then after they played the first two levels, they would answer a second

[5] http://www.libpng.org/pub/png/libpng.html
[6] https://www.freetype.org/

part of the questionnaire related to what they played. After the third level, whey would answer the rest of the questionnaire. There were questions about the experience of the players and some questions about gameplay. There were questions with open answer and likert scale answers. Along the answer of the questionnaire, we asked for some questions as a interview to take note of possible changes to the future work.

2) Results: We had 14 players to playing our protoype. The participants were between 21 and 28 years old. More than 60% of the participants considered themselves as hardcore players, and 57% play video games everyday.

After playing the first two levels, when asked if they like the song, 70% strongly agreed, and when asked whether they prefer the level with the song on the background, more than 80% strongly agreed. When asked why, they listed as reasons that the music predicted the game objects appearance, it was more fun, and without music, the level seemed longer.

We also, wrote down the score of the performance of the players, when played the level with and without the song, but the results were inconclusive, and we thought that was because of the factor that the player was unfamiliar with the gameplay and performed poorly the first level, and by the second time they knew the sequence from previous interaction with the level.

After playing the third level, when asked if they preferred the song in comparison to the previous level, they mostly agreed, and when asked if they preferred the third level comparing to the first level with song they also mostly agreed. When asked why, they stated that it was because the music was less ”happy” than the previous, and more challenging as well.

We decided to ask this question because of the difference in level design. The first level had repeating patterns and the the third had variation of patterns without repeating extensively. However, the results did not give much insights in that matter, perhaps there should been more specific questions, but such subject is out of the scope of this project.

Next we assessed the Game Experience Questions for In-Game experience from [8], as shown in Figure 8.

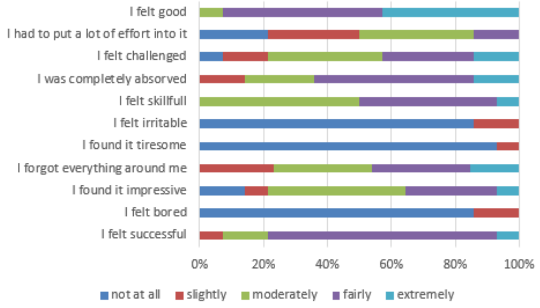


Fig. 8: Game experience questionnaire.

In reference to [8], in terms of Competence (questions 2 and 9) it can be assumed as a moderate result in skill and success, since the opportunity system was too easy to regain lives and not much unforgiving. For Immersion (question 3), this result can be related to the fact that the music and the gameplay were sightly unsyncronized, but this was because of level design, since the objects can be placed anywhere in the timeline and the creator sense of what beats of the song should be more suited to the game objects. In Flow (questions 4 and 8) there was slight sense of flow, for those who liked the songs, but because of the unsynchronization it could have affected the experience. In Challenge (questions 9 and 10) there was none, because it was easy to gain opportunities and the players did not worry losing. Negative effects (question 2 and 5) seemed to have good values, the game was not boring or tiresome, having received many comments on good aspects of the game (discussed later). Finally the Positive effects (question 11), received comments because of the sighly desincronization of the music, but also other reason that might have contributed to these values, because the first song was more "uplifting" and the second more "sad".

In terms of game engine and gameplay directions, we asked some more questions in Figure 9.

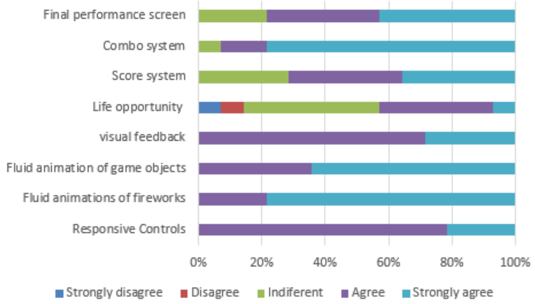


Fig. 9: Game engine and gameplay mechanics.

The animations of the fireworks and the animations of the game objects were viewed as fluid, reaching one of our goals. The visual feedback achieved moderate results. We believe it is because of the slider problem, but also because of the sight desincronization of the song and the level.

Life opportunity had the most indiferent feedback, maybe because there was no challege in losing opportunities. The players also liked the score system explanation and understood the combo system (more accuracy = more points, more combo = more points). Regarding the final scene of score performance, there were not many players that paid attention to this screen, maybe because we said at the beginning of the playtesting that the score was not important. We later acknowledged it as a flaw, since it influenced the experience.

When asked if the music and the gameplay were related, and asked how, we had many answers, as the music helps to know when the game objects would appear and when to touch them; the rhythm of the music was synchronized with the game objects; there was a choreography of fireworks and the song. However some players refer the lack of synchronization of the beats of the song and the level objects.

When asked what they liked less about our game, many referred the slider game object. It is something that needs to be worked on.

VII. Conclusion

The main purpose of this work was to develop a music rhythm game for the Android mobile platform, by creating levels with the help of an editor. Even though we found issues (such as the difficulty in placing the game objects at the right beat time, because there was no time snap ruler for the game objects, due to time constraints), we can say we accomplished other things necessary for making a game. The results show that even with the small unsynchronization, the game can still be considered a rhythm music video game. It has the main points of a music rhythm game: the player must react to visual stimuli to achieve a score and performance points in the rhythm of the song being played. However, the tests were inconclusive on whether the song would influence the gameplay, because we planned that the players would perform better if playing the level with the song instead of without song. This may be because we did not allow the player to be more familiar with the controls before testing this hypotesis. We made this test right out in the beginning, and we could have had better results if we had players playing a test level before testing what we wanted. Also, the taste of music did influence the appreciation of the level.

In terms of other subobjectives, we were able to create a game engine that implemented a fluid particle system and fluid game visual animation, as shown in the results. This implementation, created a lag-free performance for the player. The gameplay was strongly influenced by Osu!, an existing videogame.

Overall, the game experience had good results, but our use of simplicity and avoidance of challenging, in terms of opportunities of life that were easy to regain, may have affected this part of the experience.

We know that we need a lot of things to work, specialy the unsynchronization. For this we thought, in future work to create some kind of music analysis and give cues locations in time to when place the game objects in the level editor. If we accomplish this, maybe we can start working on improvements shared with the playtesters that we will take in consideration. Other issue is to remake the slider game object gameplay, because it seemed difficult to understand the way to interact with it.

Every tester agreed that they would like to play this game again, and we will focus our future work to deliver such experience to the these players and possible future players.

References

[1] Hardcore gaming 101: Otocky. http://www.hardcoregaming101.net/otocky/otocky.htm. Accessed: 2016-10-12.

[2] How do rhythm games stay in sync with the music? https://www.reddit.com/r/gamedev/comments/13y26t/how_do_rhythm_games_stay_in_sync_with_the_music/. Accessed: 2016-10-14.

[3] Industry facts - entertainment software association. http://www.theesa.com/about-esa/industry-facts/. Accessed: 2016-10-12.

[4] Is the era of music video games really over? http://mashable.com/2011/09/26/music-video-games/. Accessed: 2016-10-12.

[5] Opensl es - the standard for embedded audio acceleration. https://www.khronos.org/opensles/. Accessed: 2016-10-14.

[6] Simon turns 30: The history of the toy and gaming's first grudge. http://www.1up.com/features/simon-turns-30. Accessed: 2016-10-12.

[7] What is vsync in android? https://nayaneshguptetechstuff.wordpress.com/2014/07/01/what-is-vsyc-in-android/. Accessed: 2016-10-17.

[8] De Kort. The Game Experience Questionnaire. 2013.

[9] Mike McShaffry and David Graham. *Game Coding Complete*, chapter 7, pages 175–193. Delmar Learning, 4th edition, 2012.

[10] Martin Pichlmair and Fares Kayali. Levels of sound: On the principles of interactivity in music video games. In *Proceedings of the Digital Games Research Association 2007 Conference" Situated play*. Citeseer, 2007.